Printed: Nov 13, 2003
From: http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/conpool.html

Standard Version

Training Index

Advanced Programming for the Java 2 Platform Chapter 8 Continued: Connection Pooling

[<<BACK] [CONTENTS] [NEXT>>]

If you have used a SQL or other similar tool to connect to a database and act on the data, you probably know that getting the connection and logging in is the part that takes the most time. An application can easily spend several seconds every time it needs to establish a connection.

In releases prior to JDBC 2.0 every database session requires a new connection and login even if the previous connection and login used the same table and user account. If you are using a JDBC release prior to 2.0 and want to improve performance, you can cache JDBC connections instead.

Cached connections are kept in a runtime object pool and can be used and reused as needed by the application. One way to implement the object pool is to make a simple hashtable of connection objects. However, a more flexible way to do it is to write a wrapper JDBC `Driver` that is an intermediary between the client application and database.

The wrapper approach works particularly well in an Enterprise Bean that uses Bean-managed persistence for two reasons: 1) Only one `Driver` class is loaded per Bean, and 2) specific connection details are handled outside the Bean.

This section explains how to write a wrapper JDBC `Driver` class.

Wrapper Classes

The wrapper JDBC `Driver` created for this examples consists of the following three classes:

- `JDCConnectionDriver`
- `JDCConnectionPool`
- `JDCConnection`

Connecti n Driver

The JDCConnectionDriver.java class implements the `java.sql.Driver` interface, which provides methods to load drivers and create new database connections.

A JDCConnectionManager object is created by the application seeking a database connection. The application provides the database Uniform Resource Locator (URL) for the database, login user ID, and login password.

The JDCConnectionManager constructor does the following:

- Registers the JDCConnectionManager object with the DriverManager.
- Loads the Driver class passed to the constructor by the calling program.
- Initializes a JDCConnectionPool object for the connections with the database URL, login user ID, and login password passed to the constructor by the calling program.

```
public JDCConnectionDriver(String driver,
                String url,
                String user,
                String password)
        throws  ClassNotFoundException,
                InstantiationException,
                IllegalAccessException,
                SQLException {

  DriverManager.registerDriver(this);
  Class.forName(driver).newInstance();
  pool = new JDCConnectionPool(url, user, password);
}
```

When the calling program needs a database connection, it calls the JDCConnectionDriver.connect method, which in turn, calls the JDCConnectionPool.getConnection method.

Connecti n Pool

The JDCConnectionPool.java class makes connections available to calling program in its getConnection method. This method searches for an available connection in the connection pool. If no connection is available from the pool, a new connection is created. If a connection is available from the pool, the getConnection method leases the connection and returns it to the calling program.

```
public synchronized Connection getConnection()
         throws SQLException {

  JDCConnection c;
  for(int i = 0; i < connections.size(); i++) {
     c = (JDCConnection)connections.elementAt(i);
     if (c.lease()) {
        return c;
     }
  }

  Connection conn = DriverManager.getConnection(
                         url, user, password);
  c = new JDCConnection(conn, this);
  c.lease();
  connections.addElement(c);
  return c;
}
```

The JDCConnection.java class represents a JDBC connection in the connection pool, and is essentially a wrapper around a real JDBC connection. The JDCConnection object maintains a state flag to indicate if the connection is in use and the time the connection was taken from the pool. This time is used by the ConnectionReaper.java class to identify hanging connections.

Deadl cks and Hangs

While many client and server databases have graceful ways to handle deadlocks and hangs so you do not have to write code to handle these situations, many of the newer, lightweight distributed databases are not so well equipped. The connection pool class provides a dead connection reaper to handle such situations.

The ConnectionReaper class decides a connection is dead if the following conditions are met.

- The connection is flagged as being in use.
- The connection is older than a preset connection time out.
- The connection fails a validation check.

The validation check runs a simple SQL query over the connection to see if it throws an exception. In this example, the validation method requests the high-level description of the database tables. If a connection fails the validation test, it is closed, a new connection is initiated to the database, and added to the connection pool.

```
public boolean validate() {
  try {
     conn.getMetaData();
  }catch (Exception e) {
     return false;
  }
  return true;
}
```

Cl sing Connections

The connection is returned to the connection pool when the calling program calls the `JDCConnection.close` method in its `finally` clause.

```
public void close() throws SQLException {
  pool.returnConnection(this);
}
```

Example Application

You use a connection pool in an application in a similar way to how you would use any other JDBC driver. Here is the code for a Bean-managed RegistrationBean. This `RegistrationBean` is adapted from the auction house Enterprise JavaBeans example described in Chapters 1 - 3.

When the first `RegistrationBean` object is created, it creates one static instance of the `JDCConnectionDriver` class. This static driver object registers itself with the `DriverManager` in the `JDCConnectionDriver` constructor making it available for connection requests to all `RegistrationBean` objects created by the client application.

Passing the URL as `jdbc:jdc:jdcpool` in the `getConnection` method, lets the `DriverManager` match the `getConnection` request to the registered driver. The `DriverManager` uses simple `String` matching to find an available driver that can handle URLs in that format.

```
public class RegistrationBean implements EntityBean{

  private transient EntityContext ctx;
  public String theuser, password;
  public String   creditcard, emailaddress;
  public double balance;

//Static class instantiation
  static {
        try{
        new pool.JDCConnectionDriver(
                  "COM.cloudscape.core.JDBCDriver",
                  "jdbc:cloudscape:ejbdemo",
                  "none", "none");
        }catch(Exception e){}
  }

  public Connection getConnection()
                      throws SQLException{
        return DriverManager.getConnection(
                                 "jdbc:jdc:jdcpool");
  }
}
```

[TOP]